RL-TR-97-91
Final Technical Report
August 1997

CRISIS ACTION MESSAGE ANALYZER (CAMA) EXPLORATORY DEVELOPMENT MODEL

HRB Systems, Inc.

Mark L. Morsch and Daniel Heinze

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-91 has been reviewed and is approved for publication.

APPROVED: Sharon M. Walter

SHARON WALTER
Project Engineer

FOR THE DIRECTOR:

JOSEPH CAMERA, Technical Director
Intelligence & Reconnaissance Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
Aug 97

3. REPORT TYPE AND DATES COVERED
FINAL Sep 95 - Jul 96

4. TITLE AND SUBTITLE

CRISIS ACTION MESSAGE ANALYZER (CAMA) EXPLORATORY DEVELOPMENT MODEL

5. FUNDING NUMBERS

C - F30602-95-C-0231
PE - 62702F
PR - R536
TA - 00
WU- 02

6. AUTHOR(S)

Mark L. Morsch and Daniel Heinze

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

HRB Systems, Inc.
300 Science Park Road, Building 7
State College, PA 16804-0060

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/IRAA
32 Hangar Rd.
Rome, NY 13441-4114

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-91

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Sharon M. Walter/IRAA/4025

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Crisis Action Message Analyzer (CAMA) is a text extraction system for Army logistics teleconferencing messages. CAMA uses natural language understanding (NLU) techniques to extract the information from free text messages that is important to a logistics action officer.

14. SUBJECT TERMS

Text extraction, Natural Language Understanding, NLU, Text processing

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

Paragraph Page

1 SCOPE 1

1.1 Identification 1

1.2 System Overview 1

1.3 Document Overview 4

2 SYSTEM DESCRIPTION 5

2.1 Development Environment 6

2.2 Control Files and the Off-line System 7

2.3 The On-line System 8

3 ALGORITHMS 12

3.1 Parallel Lexical Analysis 12

3.1.1 Document Segmentation 13

3.1.2 Element Lexers 15

3.2 Shift-back Chart Parsing 18

3.2.1 The Basics of Shift-back Chart Parsers 18

3.2.2 Using Shift-back Chart Parsers 20

4 LEXICON RESEARCH 23

4.1 Overview of Lexical Source Data 23

4.2 Methodology 27

4.3 Principles 36

5 EXPERIMENT OBSERVATIONS AND EVALUATION 38

5.1 Evaluation of the Multi-phase Chart Parser 38

5.2 Comparison to Other Work 41

6 ACCOMPLISHMENTS AND CONCLUSIONS 44

7 RECOMMENDATIONS 46

8 REFERENCES 47

APPENDIX A

10 TERMS AND ABBREVIATIONS 49

LIST OF FIGURES

Figure		Page
1.2-1	CoPE Architecture	2
1.2-2	CCLS Document Processing Flow	3
2.3-1	CAMA System Menu	10
2.3-2	Netscape User Interface	11
3.1-1	Segments of a typical memorandum	13
3.1-2	Comparison of different element lexers	17
3.2-1	Maintaining the Chart	20
4.0-1	Development of the CAMA-EDM Lexicon	24
4.2-1	Sample record from merged lexicon	34
5.1-1	An example of orphaning	40

LIST OF TABLES

Table		Page
2.1-1	Software Items	6
4.2-1	Verb Complementation Patterns in SPECIALIST	32
5.1-1	Comparison of single phase and multi-phase reductions	39
5.1-2	Single phase versus multi-phase execution times	39
A-1	Acronyms and Abbreviations	49
A-2	Acronyms and Abbreviations (cont'd.)	50
A-3	Useful Terms	51

1 SCOPE

1.1 Identification

This Final Technical Report (FTR) describes the work done on the Crisis Action Message Analyzer (CAMA) Exploratory Development Model program. The CAMA system has been developed by HRB Systems and the Pennsylvania State University for Rome Laboratory/IRAA and the US Army Logistics Integration Agency. The work is being performed by HRB Systems under Government Contract #F30602-95-C-0231. The CAMA system is a single Computer Software Configuration Item (CSCI).

The format of this document has been tailored from the Data Item Description DI-MISC-80711/T for applicability to this program, with the approval of the Government.

1.2 System Overview

The Crisis Action Message Analyzer is a text extraction system for Army logistics teleconferencing messages. CAMA uses natural language understanding (NLU) techniques to extract the information from free text messages that is important to a logistics action officer. The key NLU technology used in CAMA is the L-space model for knowledge representation. L-space provides a consistent framework, based on theories in cognitive linguistics, for building interpretations of text. In addition to the L-space model, CAMA has components which tokenize and parse ASCII text documents. The ultimate goal in the development of CAMA are capabilities for automatically identifying entities and their relations, recognizing situations of interest, and establishing cause effect relationships between events.

The CAMA-EDM program is intended to redevelop the system front end, which contains the tokenizing and parsing functions, to improve robustness and maintainability, and to continue development of a lexicon. Prior to this program, a proof-of-concept system was developed. That system demonstrated the viability of the L-space model, but being a laboratory prototype, was somewhat rigid and brittle. With the CAMA-EDM program we are addressing the weaknesses of the proof-of-concept and laying the groundwork for an operational prototype. This program will produce modules which can be directly reused in an operational prototype.

The CAMA system is an application of the Computational Cognitive Linguistics System (CCLS). The term CCLS is used to refer to the underlying NLU system apart from a particular application. We will use the term CCLS when discussing the basic system technology and architecture.

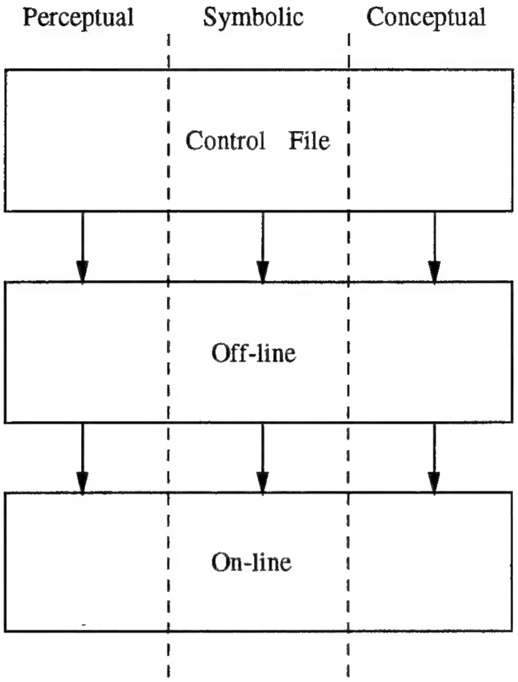


Figure 1.2-1. CoPE Architecture

CCLS has a modular design with well-defined interfaces. There are three main components (CSCs) making up CCLS: the control file system, the off-line system, and the on-line system. This organization was inspired by the architecture of the Cognitive Processing Engine (CoPE) as shown in Figure 1.2-1. The terms perceptual, symbolic, and conceptual in this figure refer to the type of information represented at each stage of processing. The intent of the CoPE architecture is to provide high run-time efficiency while retaining considerable system flexibility. In CCLS, the on-line system extracts information from documents as defined by the control files (also referred to as specification files). The specification files define the input patterns, grammar rules, and lexicon. The off-line system compiles the specification files into a binary form which is read by the on-line system. There is a clean separation between the maintainable knowledge base (the specification files) and the user application (the on-line system). There will be no source code generated when compiling the specification files, thereby no make-dependency with the on-line system.

There are four stages to the processing of documents in CCLS: unitizer, symbolizer, cognizer, and recognizer. Figure 1.2-2 illustrates these four stages. In the first stage, the unitizer accepts an ASCII document, breaks the document into segments and tokenizes each segment. The unitizer passes on a stream of tokens to the symbolizer. The symbolizer parses the token stream using a chart parser. A chart parser is driven by a set of grammar rules. When a rule matches a particular sequence of tokens, the meanings of the words are combined to produce a meaning for the entire phrase. Combining meanings of words and phrases according to syntax is called elaboration. Elaboration is the main function of the cognizer. Because elaboration is performed whenever a grammar rule is matched and typically the chart parser matches numerous rules when parsing text, the operation of the symbolizer and cognizer are interleaved. This type of interleaving allows the system to resolve syntactic ambiguity with the help of semantics. In the final stage, the recognizer detects the entities, relationships, or events that are of interest for the application. Comparing this breakdown to the CoPE architecture (Figure 1.2-1), the unitizer is perceptual, the symbolizer is symbolic, and the cognizer and recognizer are conceptual.

The CAMA-EDM program has three tasks: control and off-line development, unitizer and parser development, and lexicon research. The first two are focused on development of the system front end, which is the unitizer and symbolizer and their supporting control and off-line components. The lexicon research task will produce a core lexicon and grammar compatible with the L-space representation. This Final Technical Report describes the important algorithms written for the two development tasks as well the results of the lexicon research task. This report includes a comparison of CAMA to other information extraction systems, an overview of the major accomplishments of the program, and recommendations for future directions.

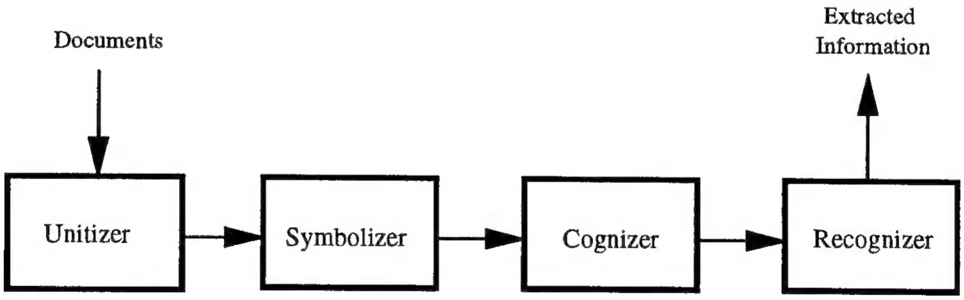


Figure 1.2-2. CCLS Document Processing Flow

1.3 Document Overview

Section 1 contains background and overview information relating to the program, the system, and this document. Section 2 contains a description of the CAMA system and an overview of hardware and software environment. Section 3 describes the important algorithms developed for lexical analysis and parsing under the CAMA-EDM program. Section 4 presents the results of the lexicon research including the sources of lexical data and the methodology for developing a lexicon for use with logistics teleconferencing messages. Section 5 discusses the experiments run with the system and how the system compares with other efforts. Section 6 summarizes the important accomplishments and conclusions of the program. Section 7 gives recommendations for possible application and extension of this work. Section 8 lists the references used over the course of the program. Appendix A contains notes relevant to this document and program.

2 SYSTEM DESCRIPTION

Throughout the CAMA-EDM program, our emphasis has been the development of a system which performs with the necessary accuracy but retains a high level of flexibility. To reach these goals, we have designed the system according to a well defined architecture. The architecture was inspired by CoPE, as described in Section 1.2. This architecture motivates the breakdown of CAMA into the control file system, the off-line system, and the on-line system. Much of the system's flexibility comes from the use of control files. Each phase of processing within CAMA can be programmed through control files. This allows a user to adapt the system to new and changed message formats and tune the system to improve accuracy. These changes do not require a recompilation of any source code. The control files are converted into binary objects by the off-line system. The on-line system reads the binary objects and uses them to guide the extraction process.

The CAMA system is designed to operate as a standalone demonstration system. The system is configured to read teleconferencing messages stored in plain ASCII files with one message per file. For each teleconferencing message processed, the system produces four output files. These output files illustrate partial results as well as a summary of the extracted information.

The CAMA-EDM program has focused on development of the system front end. The front end consists of two software modules, the unitizer and the symbolizer. The extraction capabilities of the current CAMA system, for the most part, come from the actions of these two modules. Given this, the extraction observed will be almost exclusively pattern-based. However, the system has been designed to support a text understanding capability which, when completed, will be provided by the cognizer and recognizer modules. Our goal is the eventual integration of a mature L-space functionality which would support cause-effect tracing and historical archiving. Supporting these goals has guided the design of the unitizer and symbolizer and has motivated the development of two technologies: parallel lexical analysis and shift-back chart parsing. These technologies are described in detail in Sections 3.1 and 3.2, respectively. Therefore, while the CAMA system cannot be viewed as complete, its current level of functionality does provide a useful and demonstrable capability.

In this section we profile the methodologies used in development and describe the main components of the system. Section 2.1 discusses the development environment, describing the hardware platform, software tools, and engineering methodology. Section 2.2 describes the control file system and the off-line system. Section 2.3 describes the online system, driver script, and Netscape user interface.

2.1 Development Environment

The software items used in development are listed in Table 2.1-1. All software items used in development are HRB capital, and as such will not be delivered. The only items necessary for running the system are the CAMA binary executables and shell scripts. All run-time vendor libraries will be statically linked into the executables. The CAMA system will run on any Sun-4 architecture with an operating system of Solaris 2.0 or higher.

Item	Vers.	Mfgr.	Notes
C++ Compiler	4.0.1	SunSoft	Sun C++ compiler
Tools.h++	6.0.4	Rogue Wave Software	Class library bundled with Sun C++
Yacc++	2.0	Compiler Resources	Parser generator tool
RCS	5.6	FSF	Gnu configuration management tool
Keeper	1.1	HRB	Distributed configuration management tool wrapping RCS
Netscape Navigator	2.0	Netscape Communications Corporation	Used to develop rapid prototype of user interface

Table 2.1-1. Software Items

The CAMA system has been developed from the ground up using an object-oriented methodology. Under previous programs, a level of functionality had been developed which demonstrated the viability of the L-space model. The CAMA-EDM program has built the infrastructure, in particular the unitizer and symbolizer, to support this model. An object-oriented approach was taken in order to produce a robust, maintainable infrastructure. This has resulted in modular source code with well defined interfaces.

The overall quality of the system has been enhanced by exercising software engineering discipline throughout development. This discipline is seen in the clean system organization and in the enforcement of configuration management policies. These policies maintained a single consistent baseline across multiple developers throughout the program. We also used a commercial C++ class library in the development of the off-line and on-line system. Using these standard classes facilitates the development of sophisticated data structures and algorithms while maintaining robustness.

Each software unit has been tested standalone, as well as integrated with the overall system. The standalone tests included writing driver and stub code which simulated unit-to-unit interfaces. We also created significant test data to exercise both the normal system behavior and potential error conditions. To support these tests and make the system more user-friendly, extensive error checking and handling was designed into the off-line system. The integrated system was tested on the nearly 200 unclassified telelconferencing messages furnished by the Army. The Software Test Plan [13] contains descriptions of the system-level tests. Together, this testing has proven the system to be quite stable and robust.

2.2 Control Files and the Off-line System

Control files, also referred to as specification files, provide the user with the capability to program the system. There are three types of control files in the CAMA system: unitizer control files, symbolizer control files, and L-space control files. Corresponding to the three types of control files, are three off-line programs: unitgen, symgen, and lsgen. These programs convert the control files into binary objects which are read in by the on-line system. This section gives a brief overview of the control files and the off-line system. These are described in detail in the CAMA-EDM Software User's Manual [14].

Unitizer control files specify how a document is segmented, tokenized, and identified. There are three types of structures making up a unitizer control file: element definitions, segment definitions, and document definitions. Elements are the lowest level structures, corresponding to individual words or groups of words. Segments are mid-level structures matching portions of a text and are composed of a sequence of elements. Documents are the top level structure, covering the entire text. Definitions of the same type cannot be nested within one another. That means that elements cannot be within other elements, segments cannot be within other segments, etc. The three levels of specification are intended to correspond to the natural organization of many types of documents. This organization makes easier the specification of unitizer control files.

Symbolizer control files specify how the free text portions of a document are parsed. A symbolizer control file contains a list of declarations followed by a set of grammar rules. There are three types of declarations: terminals, nonterminals, and functions. The terminal declarations define a set of token names which appear only on the right hand side of grammar rules. The nonterminal declarations define a set of token names which can appear either on the left hand side or the right hand side of grammar rules. The terminals can be viewed as the input token types. The nonterminals represent the token types which are produced by the execution of the rules. The function declarations specify a set of function names which can be referenced on the right hand side of a rule. Each grammar rule specifies a sequence of terminals or nonterminals which reduce to a particular nonterminal. The function included with a grammar rule is executed when the sequence of tokens on the right hand side is matched. One symbolizer control file makes up one phase of a parser. This method of specifying the behavior of the parser supports pattern matching based on word part-of-speech and is compatible with multi-phasing (see Section 5.1).

The off-line system consists of executable programs and shell scripts. The unitgen program compiles the unitizer control files, the symgen program compiles the symbolizer control files, and the lsgen program compiles the L-space control files. The error messages produced by the unitgen and symgen programs are described in the Software User's Manual [14]. Note that because of the emphasis in the CAMA-EDM program on development of the front end, the L-space control files are not user maintainable. Corresponding to the three programs are three shell scripts: ubuild, sbuild, and lbuild. These scripts support the creation and maintenance of a system configuration. The system configuration for CAMA consists of the control files to be used as input to the off-line compilers. A fourth script, camagen, controls execution of the other programs and scripts making up the off-line system. To properly build the binary form of the control files, the off-line compilers must be executed in a particular order. The order is symgen first, lsgen second, and unitgen third. The camagen script automatically ensures that this order is followed.

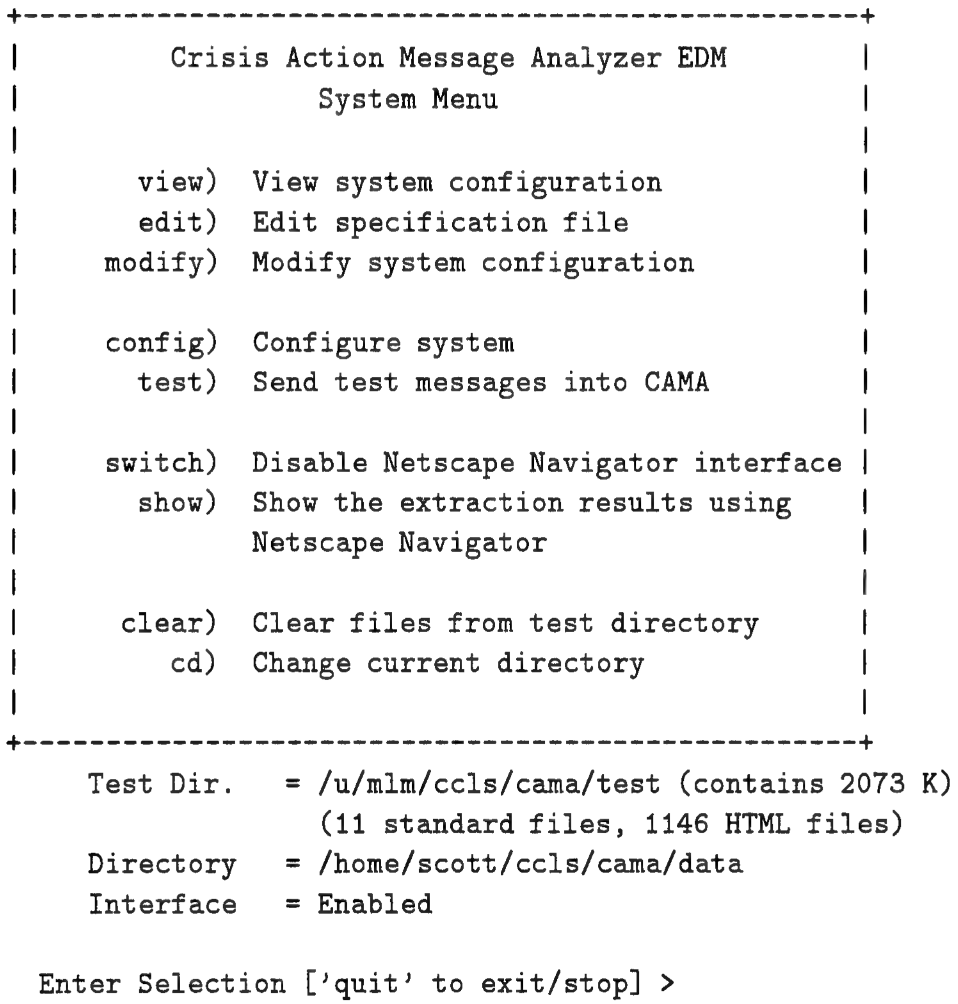
2.3 The On-line System

The online system contains an executable program, cama, and the script, CAMAdriver. The CAMA system is operated using a few, simple commands provided by the driver script. Figure 2.3-1 shows the menu which appears when running the CAMAdriver script. The default starting directory for the CAMAdriver script is `$CAMA_HOME/data`, indicated by the line starting with `"Directory    ="`. For the menu shown in Figure 2.3-1, $CAMA_HOME is set to `/u/mlm/ccls/cama`. The current directory is the source of test messages. New messages to be processed should be placed in this directory. The test directory indicates where the files generated by the on-line system will be stored. This directory is always `$CAMA_HOME/test`. Within the system menu box, there are nine commands listed. The first three commands, `view`, `edit`, and `modify`, allow manipulation of the system configuration and control files. The next two commands, `config` and `test`, execute the off-line and on-line systems, respectively. The commands `switch` and `show` control the Netscape user interface. The last two commands are `clear` and `cd`. The `clear` command deletes all files in the test directory. The `cd` command changes the current directory.

Sending messages through the on-line system is done using the `test` command. The driver script looks for messages in the `$CAMA_HOME/data` directory by default. The user can enter a single message file name or can use a wildcard to match several of the message files. After accepting the message file name(s), the system processes the message (or messages) and produces four output files per message. These output files have the original message file name appended with the extensions .seg, .tok, .sym, and .data. So for the file tlcf_1_01.msg, the following set of files would be produced: tlcf_1_01.msg.seg, tlcf_1_01.msg.tok, tlcf_1_01.msg.sym, and tlcf_1_01.msg.data. These output files, along with a copy of the original message file, are put in the $CAMA_HOME/test directory. The file with the .seg extension contains a segmented form of the message. The file with the .tok extension contains a tokenized form of the message. Both of these files are produced during unitization. The file with the .sym extension contains a parsed form of the message. This file is produced by the symbolizer. A summary of the extracted data is in the file with the .data extension. This file can be considered the final system output.

The Netscape Navigator interface allows a user to view a message at different stages of processing. This interface consists of an interlinked series of Web pages which present annotated forms of the original messages. These pages are interactive, allowing a user to highlight particular segments, elements, and patterns. This interface can be viewed as both a visual means of inspecting internal processing of the on-line system as well as a prototype of an operator's interface. Figure 2.3-2 shows the starting page of the interface. This page consists of two frames. At the top is a command bar which provides links the home page, help, and general information. The lower frame welcomes the user to the CAMA-EDM Extraction Interface and lists the messages for which results can be viewed.

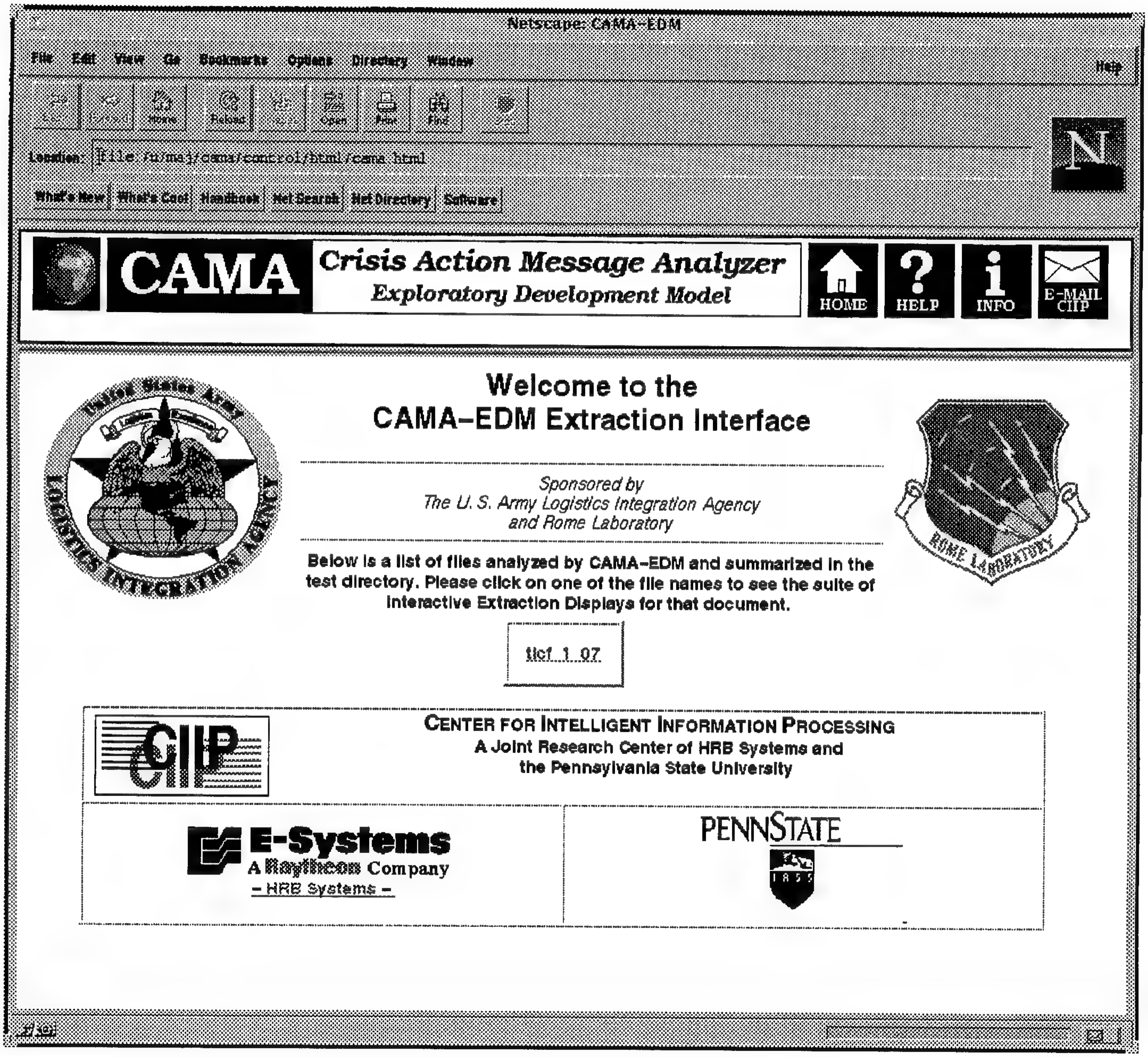
Netscape is spawned from the driver script using the show command. The files which hold the HTML pages for the interface are generated automatically by the on-line system. Generating the numerous files required does slow message processing, so the driver script has the switch command to suppress generation of the HTML pages. The current state of the interface switch is indicated by the "Interface =" line at the bottom of command menu. When the interface is disabled the show command is not displayed. With the interface disabled, results can be viewed using the four output data files described previously. Although the same information is represented in these four data files, the Netscape interface gives a superior visual presentation.



```
+-----------------------------------------------------+
|         Crisis Action Message Analyzer EDM          |
|                  System Menu                        |
|                                                     |
|       view)  View system configuration              |
|       edit)  Edit specification file                |
|     modify)  Modify system configuration            |
|                                                     |
|     config)  Configure system                       |
|       test)  Send test messages into CAMA           |
|                                                     |
|     switch)  Disable Netscape Navigator interface   |
|       show)  Show the extraction results using      |
|              Netscape Navigator                     |
|                                                     |
|      clear)  Clear files from test directory        |
|         cd)  Change current directory               |
|                                                     |
+-----------------------------------------------------+
     Test Dir.    = /u/mlm/ccls/cama/test (contains 2073 K)
                    (11 standard files, 1146 HTML files)
     Directory    = /home/scott/ccls/cama/data
     Interface    = Enabled

  Enter Selection ['quit' to exit/stop] >
```

Figure 2.3-1. CAMA System Menu


Netscape: CAMA-EDM
File Edit View Go Bookmarks Options Directory Window Help
Home Reload Open Print Find
Location: file:/u/maj/cama/control/html/cama.html
What's New What's Cool Handbook Net Search Net Directory Software
CAMA
HOME
HELP
INFO
E-MAIL CIIP
United States Army Logistics Integration Agency
Rome Laboratory
CIIP
E-Systems A Raytheon Company
PENNSTATE
1855

Crisis Action Message Analyzer
Exploratory Development Model

Welcome to the
CAMA–EDM Extraction Interface

Sponsored by
The U. S. Army Logistics Integration Agency
and Rome Laboratory

Below is a list of files analyzed by CAMA-EDM and summarized in the test directory. Please click on one of the file names to see the suite of interactive Extraction Displays for that document.

ticf_1_07

Center for Intelligent Information Processing
A Joint Research Center of HRB Systems and
the Pennsylvania State University

– HRB Systems –

Figure 2.3-2. Netscape User Interface

3 ALGORITHMS

The main system components developed under the CAMA-EDM program were the unitizer and the symbolizer. The unitizer performs text segmentation, tokenization, and document identification. The symbolizer parses the free text portions of a document. Both of these modules were designed to be integrated with a future L-space module. The most important capability necessary to integration is the preservation of ambiguity through all phases of processing. This will allow an L-space module, which will be guided by semantics, to have access to the information needed in order to make the right decisions. Preserving ambiguity requires that the unitizer and symbolizer create and maintain different, but still valid, representations for the same portions of text. The challenging aspect of this problem is to develop algorithms which satisfy this requirement, but are also efficient, robust, and programmable.

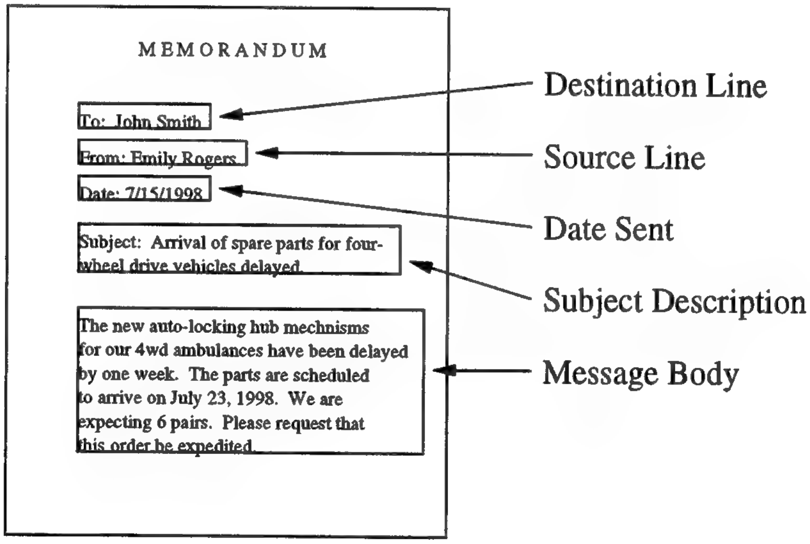
These requirements have motivated the development of algorithms in two technical areas: lexical analysis and free text parsing. In the area of parsing, previous work on CAMA had resulted in a highly efficient approach to parsing, called shift-back chart parsing. To satisfy the input requirements of the chart parser, a new method of parallel lexical analysis, also referred to as chart lexing, was developed under CAMA-EDM. Parallel lexical analysis and shift-back chart parsing are the key technologies behind the unitizer and symbolizer. In Section 3.1, we discuss parallel lexical analysis, and in Section 3.2 we discuss shift-back chart parsing.

3.1 Parallel Lexical Analysis

The unitizer produces tokens using parallel lexical analysis. There are several differences between this approach and typical lexical analyzers constructed using the programs lex or flex. First, no source code is generated when compiling the unitizer control files. Output is in the form of binary objects. Second, the unitizer control files are specified in an object-oriented way, with structures corresponding to the natural organization of documents. The third difference is that the unitizer approaches lexical analysis partly from the top down, recognizing the major segments of a document first. The typical approach to lexical analysis is entirely bottom-up. The fourth difference is the use of multiple lexer objects, all running in parallel. When the unitizer is running, there are many small lexers all executing at once. The name parallel lexical analysis is referring to this characteristic but also includes all of the differences described here. In this section, we describe the algorithms involved in doing document segmentation and execution of the lexer objects.

3.1.1 Document Segmentation

A segment is a portion of a document, spanning from a part of a line up to several lines, identifiable via structural delimiters. Segments are intended to correspond to the natural divisions of a text. The segments for a typical memorandum are shown in Figure 3.1-1. If a portion of a document can clearly be recognized as separate, based on the the document structure, then that portion will most likely make a reasonable segment. A segment provides a context which guides element matching and tokenization. An element defines the pattern of a single lexeme or token and is discussed in detail in the next section. For example, since a date can be expected after the word "Date:" in a memorandum, this helps focus the operation of the unitizer. Only the element lexers which match dates need to be active while scanning the segment. This is a simple illustration of how context guides the tokenization process.



MEMORANDUM

To: John Smith

From: Emily Rogers

Date: 7/15/1998

Subject: Arrival of spare parts for four-wheel drive vehicles delayed.

The new auto-locking hub mechnisms for our 4wd ambulances have been delayed by one week. The parts are scheduled to arrive on July 23, 1998. We are expecting 6 pairs. Please request that this order be expedited.

Figure 3.1-1. Segments of a typical memorandum

The segmenter function within the unitizer recognizes segments of the type shown in Figure 3.1-1. The segmenter is the first function run on a document within the unitizer. Segments of a document are located by scanning the text for the segment delimiters. These delimiters are regular expressions. The algorithm for finding the segments first scans the documents for all possible beginnings of segments. This forms a list of candidate segments, sorted according to the starting point of the segment delimiters. The list is then traversed, and the algorithm attempts to find the end of each candidate segment. As this is done, segments which overlap with other segments or have no end point are deleted from the list. The strategy used here favors shorter segments. So if one segment is completely contained within another, the larger is deleted. The strategy also favors segments with earlier starting points. If two segments overlap, then the one with the later starting point is deleted. In this algorithm, delimiters from two different segments are allowed to overlap, but the body of the segments may not. This algorithm produces a list of nonoverlapping segments which can then be tokenized and parsed.

Like all of the main functions within the on-line system, segmentation is programmed using control files. The format of these control files is described in detail in the Software User's Manual [14]. Here, we highlight the important features. The definition of a segment contains a list of possible beginning and ending regular expressions. It also contains a pattern. This pattern specifies a list of particular elements and strings which must appear in the specified sequence to get the correct tokenization of the segment. The pattern provides the means to guide the tokenization process. For example, a segment which matches the date sent in the memorandum shown in Figure 3.1-1 would have a pattern consisting of an element for matching a date. Only text which matches the date element would be correctly interpreted in that context. Conversely, the body of the memorandum is free text. The segment matching the message body would not limit the types of elements which could be matched. This flexibility in limiting context allows segments to accurately interpret free as well as formatted text.

After segmenting a text file, there may be portions of the text which do not fall into any segment. These segments are handled by identifying the type of document. Within the unitizer, a document is some ordering of segments which are characteristic of a particular document type. A document is the highest level structure matched by the unitizer. Documents are intended to cover an entire file, but they can be specified to match portions of a text. A document is defined by the segments which make it up. So if two text files have the same number and types of segments, then those two files will be identified as the same document type. This allows handling of different document types by the same set of specifications.

Documents are specified in control files in a way similar to segments. A list of segments defines the document. Included in the definition of a document, is an expression declaring the default segment. The default segment is the segment type assigned to the portions of text not covered by any other segment. In order to assign default segments, a document identification must be made. So, within the unitizer, after a text file is segmented, the list of segments is matched against the defined set of document types. If there is a match, then the portions of text not within any segment are defined as default segments. That text is tokenized according to the definition of the default segment. If the default segment is nil or not included in the document definition, then the text not within any segment is ignored during tokenization and parsing.

3.1.2 Element Lexers

The lowest level structures matched by the unitizer are elements. An element matches a sequence characters, which may include white space, representing a distinct lexical unit. Elements can be considered the building blocks of a text. Each is, in a sense, indivisible. The sequence of characters which matches an element is called a token, and each token is assigned a definition, or set of possible definitions. The process of matching elements is similar to the process used in lexical analyzers built using lex or flex. In both processes, characters are grouped together to form lexemes. However in the unitizer, elements are matched by a set of small element lexers. Each lexer matches one particular type of element. The typical flex lexer has a single function which matches the specified patterns. This prevents a sequence of characters from being matched in more than one way. Using element lexers, the same set of characters can make up more than one lexeme. The other critical difference in element lexing, is that the number and type of element lexers active at any one time is determined by the current segment pattern. Doing context-sensitive lexing is difficult using flex or lex.

As we have said, an element lexer matches a particular type of element. Just like segments and documents, an element is specified in a control file. A complete description of element specifications can be found in the Software User's Manual [14]. Elements are defined using expressions and patterns. An expression is a short sequence of characters of a particular type. Expressions can match integers, real numbers, strings, or free text. Expressions may have a defined length and may be limited to an enumerated set of values. Elements are defined by a sequence of expressions which form a pattern. Each element definition contains one and only one pattern. A typical control file will contain a few dozen element definitions.

Elements are most useful for matching items such as dates, times, coordinates, or any distinct item which has a well defined, regular format. An element should typically be short, less than 40-50 characters, and should not cross over a line break, except in cases where the line break matches embedded white space. Elements are self-contained and independent. The scope of an expression is limited to the element in which it is defined. This means that two identical expressions can appear in two different element definitions. Likewise, the pattern, in which expressions are referred to by name, can only reference expressions defined within their element. Having no dependency between element specifications supports the creation of an independent object for matching each element in text.

The key advantage to lexical analysis based on element lexers, is the capability to have several lexers active at one time. This allows the same piece of text to be interpreted in more than one way. Figure 3.1-2 shows a comparison of three element lexers.

Figure 3.1-2 shows the tokens produced by three element lexers for the text "on July 23, 1998". We will assume for this example that there are only three element lexers active. The top box shows a free text lexer. A free text element will create a separate token for each white delimited string and for certain punctuation marks. The arrows in the diagram indicate the current position of the lexer. After the free text lexer has processed the text, five tokens have been created. A token is indicated by a rectangle drawn around a piece of text. The middle box illustrates the actions of a number lexer. For this text, the lexer creates two tokens, matching "23" and "1988". The other text is ignored by the lexer. The bottom box shows a date lexer. This lexer creates a single token matching the string "July 23, 1998". The element defining this lexer is specified to match a date in the format shown.

After a segment has been tokenized using the three element lexers, all of the tokens produced are combined to form a single chart. That chart is then passed onto the symbolizer which does grammatical analysis using chart parsing. In this example, the single token matched by the date lexer for the text "July 23, 1998" fits best in context. However, the token for the word "on" was produced by the free text lexer. This simple example shows that combining the results of these lexers to build a single chart is necessary to support chart parsing.

A concept fundamental to CCLS and CAMA, is deferring interpretation of text until the broader context can be introduced. So in lexical analysis, we do not want to make a decision which can better be made in parsing. That is why we have chosen a strategy which will likely produce redundant tokens in certain situations. For example, the two tokens produced by the number lexer in Figure 3.1-2 will not contribute to the final interpretation. However, if the text had read "on July 23, 1998 soldiers", then it is not as clear that the "1998" should be interpreted as a number modifying the word "soldiers" or as part of the date "July 23, 1998". In this case, having both tokens available to the chart parser is necessary to support the correct interpretation, if it can be uniquely determined.

While using element lexers does produce redundant tokens, this can be reduced by using the context of the current segment. For example, if the date sent line were a separate segment, see Figure 3.1-1, then the types of lexers active for tokenizing this segment can be limited based on the format of the information expected. For the date sent line, a single element lexer matching the format "MM/DD/YY" (MM = month, DD = day, YY = year) would be sufficient. The unitizer spawns element lexers based on the elements specified in the segment pattern. This capability leads to highly efficient tokenization and parsing of formatted fields within messages.

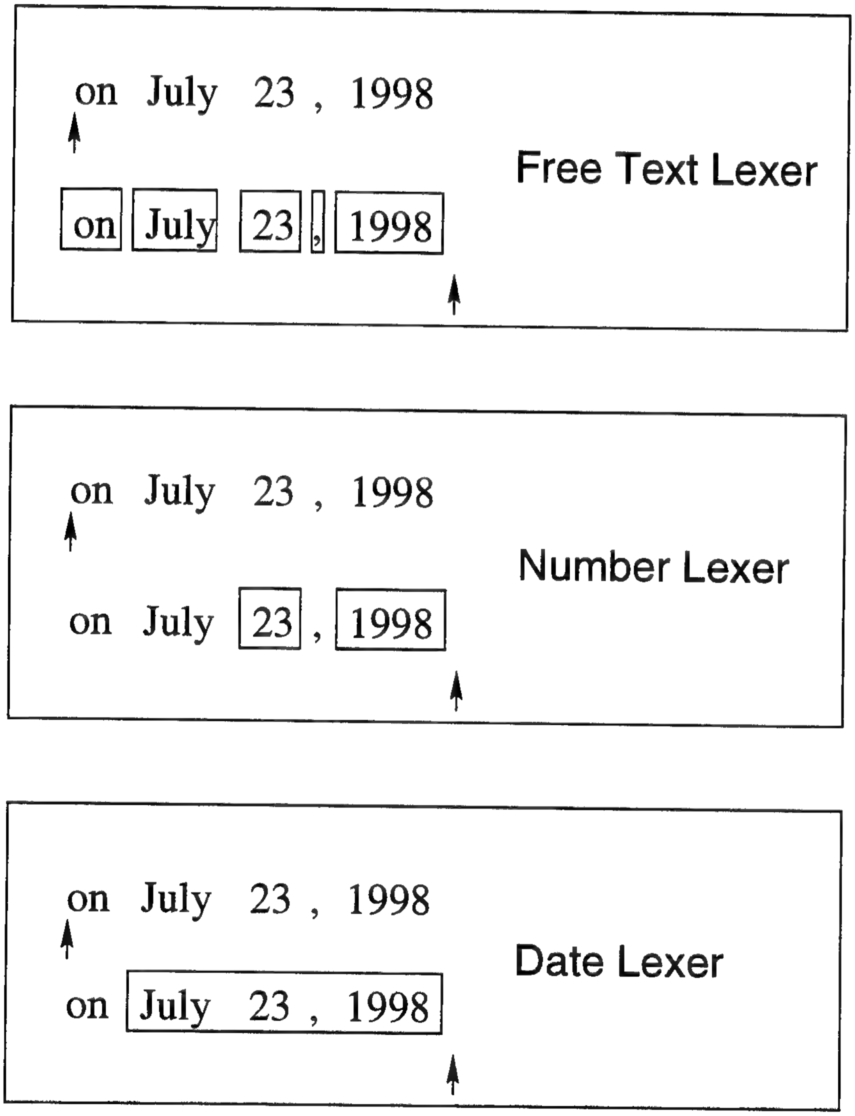


Figure 3.1-2. Comparison of different element lexers

3.2 Shift-back Chart Parsing

In this section we give an overview of shift-back chart parsing and its application to natural language processing systems. A shift-back chart parser is different than other parsers that maintain a chart in that they employ the shift-back operation to efficiently search all shift-reduce and reduce-reduce conflicts. Section 3.2.1 is devoted to the basics of chart parsing and the shift-back mechanism. In Section 3.2.2 the use of a shift-back chart parser in natural language processing systems is discussed.

3.2.1 The Basics of Shift-back Chart Parsers

3.2.1.1 Grammar Rules

As with all parsers a shift-back chart parser has two basic actions, shift and reduce. Given a linearly ordered set of tokens, a parser must inspect each in order to ascertain whether or not the token being inspected is part of an acceptable pattern. The grammar specifies what this pattern must be through a series of *productions* or grammar rules of the form $L \rightarrow R_1 R_2 ... R_n$ where L and the R_i are all token types. When the parser accepts another token as part of a pattern it is trying to match, it is said to have been *shifted* on the stack. When the pattern is fully matched (all the tokens on the stack have the appropriate type in the appropriate order) then a new token is created with type L. This action is called a *reduction.* If the parser could either shift or reduce given the state of the stack and the type of the token it is currently looking at, then we say it has a *shift-reduce* conflict. If it matches two different rules within the same string of tokens then it is said to have a *reduce-reduce* conflict. Rather than arbitrarily decide which action to take, a chart parser will try both actions, recording the result of each choice in a chart.

3.2.1.2 Tables

Shift-back chart parsers only accept context free grammars. However, they can and do accept finitely ambiguous grammars. Context free grammars with productions of the form $S \rightarrow S$ are infinitely ambiguous (besides being useless) and will cause a shift-back chart parser to go into an infinite loop. Such grammars are easily detected in off line processes as tables are being generated from the grammar specification. One peculiarity of a shift-back chart parser is that the tables generated for one are not reduced in any way as with LR$_n$ or LALR tables. They rather are represented in tree form which is represented in a condensed form of a sparse matrix. This makes shift-back chart parsers theoretically inefficient but makes their construction and tables exceedingly simple. In natural language processing applications they are highly efficient and easy to maintain.

3.2.1.3 The Chart

Since chart parsers parse ambiguous grammars it is often the case that shift-reduce and reduce-reduce conflicts arise. When this happens, both actions must be tried since there is no *a priori* structure that an ambiguous grammar has that allows anything but an arbitrary choice. The potential exists (and is easy to realize) that some patterns can be found and matched multiple times. To avoid this, a record of all reductions is kept on a *chart.* A chart is simply a data structure that allows the parser to avoid duplication of effort.

On a shift-back chart parser, the chart can be thought of as a matrix. The top row of the matrix is filled from left to right with the input tokens. The parser inspects each token in the chart, starting on the left and moving to the right, searching for patterns to match. When a pattern is found then a new token is created with the type specified on the left hand side of the production matched. This new token is placed on the chart underneath the first token in the string of tokens making up the pattern matched in the chart. Figure 3.2-1 illustrates this with a series of charts and the actions taken by the chart parser because of the state of the chart. The dot, •, is on the left of the token that the parser is currently looking at. After the N and V tokens have been shifted on to the stack (charts 1 and 2), a pattern is matched and a new token with type S is placed underneath the N token (chart 3). The first token in the pattern is also marked so that the parser never again tries to match a pattern starting with that token. Instead, after shifting back (charts 4 and 5), the parser moves down the column to find an alternative token. The parser then looks at the S token instead of looking at the N token (chart 6). However, since no production starts with an S token, it does not get shifted onto the stack. Since there are no other tokens to look at, the parser stops.

3.2.1.4 The Shift-back Action

In a shift-back chart parser, the parser resolves reduce-reduce conflicts simply by recording both reductions in the chart. To resolve shift-reduce conflicts, it always shifts first and then reduces later. Also, since more than one token can appear in a column, the parser must try one shift and then the other at a later time. Hence, at some time

Grammar Rule: $S \rightarrow NV$

chart		action
1	•N V	shift N
2	N •V	shift V
3	N V • S	reduce $NV \leftarrow S$
4	N •V S	shift-back
5	N V •S	shift-back
6	N V S	stop

Figure 3.2-1. Maintaining the Chart

it must come back to the column it shifted from to try other alternative shift or reduce actions. The mechanism for doing this is the *shift-back* action. Essentially, the shift-back action "unshifts" a token and tries the next one in the same column. When it comes to the bottom of the column then it has to shift-back the previous token. When it runs out of columns then it is done parsing. By this simple means all possible parses of a string may be found.

3.2.2 Using Shift-back Chart Parsers

3.2.2.1 Completeness and Correctness

It is a natural question to ask if a parser matches all the patterns to be found in any string of tokens and only those patterns that are found in such a string. The first question is one of *completeness* and the second one of *correctness.* Shift-back chart parsers are indeed both complete and correct. Given a context free grammar without infinite ambiguities, they detect all and only the patterns specified in the grammar. The correctness is guaranteed by the fact that we shift only when a pattern could be matched and completeness is guaranteed by the fact that we ultimately try all possible combinations through the shift-back mechanism.

3.2.2.2 Efficiency

The asymptotic space and time complexity of a shift-back chart parser are both exponential in the size of the input. However, it is also the case that if the average length of productions is three or less that the average time and space complexity is quite manageable, being on the order of cubic in the input size. Efficiency is especially boosted with respect to both space and time by the intelligent use to the chart. Multiple parses can be encoded, using the chart, into a packed parse forest which efficiently records all the parses found by the chart parser.

3.2.2.3 Reduction Code

It is usually the case that the purpose of finding a pattern is not just to create a new token. Some action needs to be taken. This is the purpose of *reduction code.* Reduction code is a set of instructions that tells the parser what to do when it finds a pattern. A necessary part of the reduction code of any chart parser is to create a new token and place it correctly in the chart. However, information associated with each token of the pattern matched may allow us to eliminate the pattern as a meaningful one. For example, the sentence "When he came in the temperature dropped" tells of the entrance of someone with a chilly personality. In the middle of the sentence is the prepositional phrase "in the temperature" which a chart parser would match as a valid pattern. However, it would be rejected by intelligent reduction code since it makes no sense. Through semantic analysis such as this, the chart can be purged of many unnecessary and meaningless tokens.

3.2.2.4 NLP Systems

Natural language is full of ambiguities and subtleties that cannot be completely resolved with pattern matching techniques alone. However, the correct use of chart parsers can go a long way toward resolving these problems. First, since natural language is so ambiguous in its syntax (grammar), chart parsers are a natural means of detecting the various possible ways a sentence is to be parsed. Second, the syntax of natural languages lends itself to decomposition into small productions which makes shift-back chart parsers particularly efficient. Third, the syntax of natural languages can also be decomposed into natural stages, especially constituency based languages like English. In English, for example, there is a natural phrase structure that allows its grammar rules to be expressed in terms of smaller phrases that can be combined into larger ones. English sentences can therefore be parsed in multiple stages. Experience has shown that connecting shift-back chart parsers in multiple phases decreases both the average time and space complexity of parsing very ambiguous English sentences without significantly degrading the performance of parsing unambiguous sentences. Finally, by interleaving semantic analysis with syntactic (grammatical) analysis, the average time and space complexity of natural language analysis is dramatically reduced. In fact, it is often the case, especially when dealing with sentences with many prepositional phrases, that from the thousands of possible syntactic interpretations, a unique correct interpretation can be found without an exhaustive search.

4 LEXICON RESEARCH

Lexicon research during the CAMA-EDM program has focused on the investigation and application of existing lexicons for use with the CAMA front end. Because the front end modules are concerned with tokenization and syntax, our primary objective has been to build a lexicon that is suitable for these processes. We have, however, also been concerned with the semantic aspects of the lexicon that are needed for the back end of CCLS.

Existing lexicons have been used to as great a degree as possible. The task of building lexicons is very time consuming, and so it is in the best interest of the CAMA-EDM program that we leverage previous work. We have investigated a number of sources for lexicons and are currently using two of these sources in addition to lexicon development that was done on previous CAMA programs.

The development of the CAMA-EDM lexicon is illustrated in Figure 4.0-1. In the following sections, we will note each of the sources and provide a detailed description of the sources that are being used. For each of the sources being used we will describe the methodology involved in selecting and merging data into a single CAMA-EDM lexicon. Finally, we will discuss some principles of lexicon development that relate to the semantic aspects of the lexicon.

4.1 Overview of Lexical Source Data

There are a number of sources for lexical data. These vary in coverage, depth and focus as well as availability and cost. We have investigated the following sources:

- Linguistic Data Consortium at University of Pennsylvania
- Language Representation Database at University of Delaware
- The Natural Language Software Registry at the German Research Center for Artificial Intelligence (DFKI)
- Sharable Ontologies Library at Stanford University
- Microkosmos at New Mexico State University and Carnegie Mellon University
- Cyc from Cycorp, Inc.

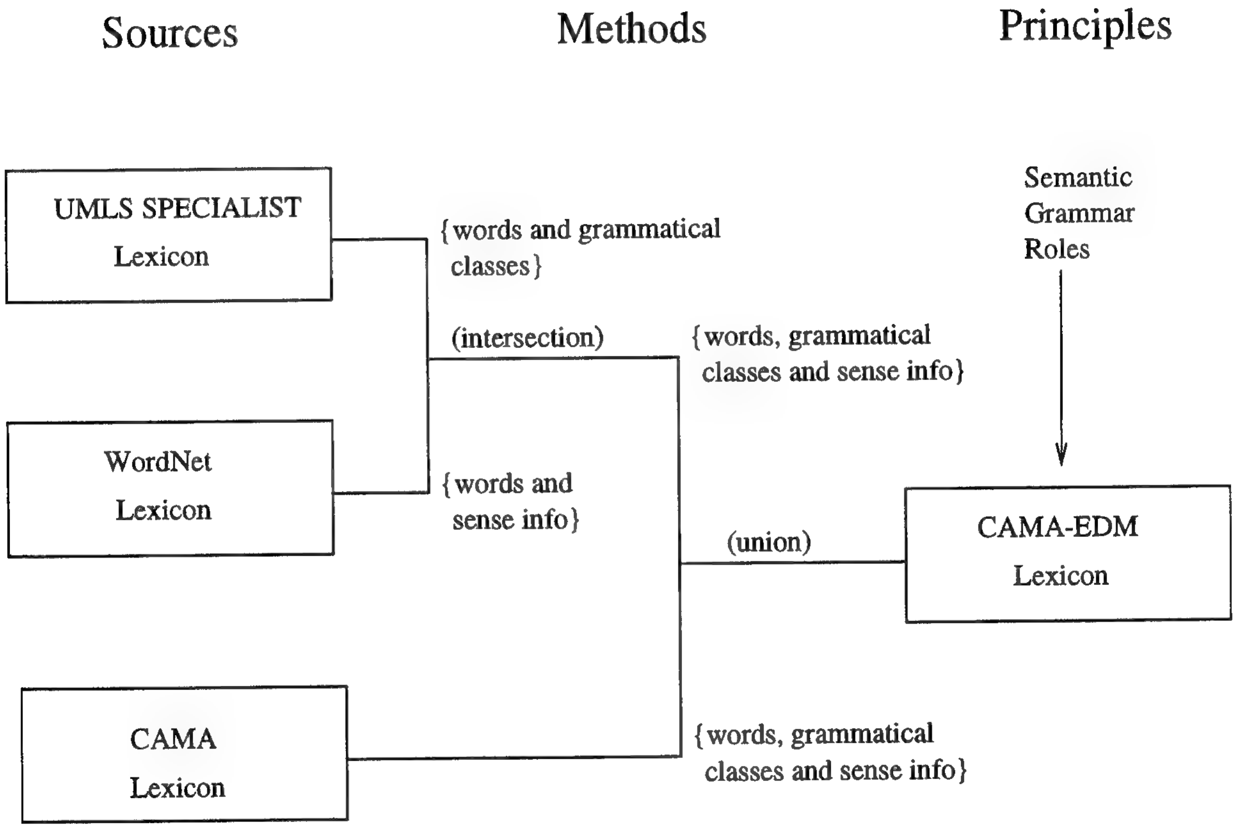


Figure 4.0-1. Development of the CAMA-EDM Lexicon

- The Moby Database
- WordNet from Princeton University
- Unified Medical Language System (UMLS) from the National Library of Medicine

The Linguistic Data Consortium (LDC) is an open consortium of universities, companies and government research laboratories. It creates, collects and distributes speech and text databases, lexicons, and other resources for research and development purposes. The University of Pennsylvania is the LDC's host institution. The LDC was founded in 1992 with a grant from ARPA and is partly supported by the National Science Foundation. LDC organizes its holdings as wideband speech corpora, telephone speech corpora, text corpora, and lexical databases. Holdings within the lexical databases are CLEX and COMLEX.

CLEX is from the Dutch Centre for Lexical Information at the University of Mijmegn and the Max Planck Institute for Psycholinguistics. CLEX contains lexical information for Dutch, German and English. Each database contains orthography, phonology, morphology, syntax and frequency information.

COMLEX is from the LDC and is a 90,696 word dictionary of English pronunciation information for use in speech recognition.

Language Representation Database is a current research project which is attempting to develop a method for merging information from WordNet, the Brown Corpus (a collection of English text - small and somewhat dated by current corpus standards) with frequency, phonetic, syllabic and morphological information in order to form a small, customized lexicon. As yet, the interfaces to the various lexical sources is not fully operable, and the development of custom lexicons is a future research effort.

The **Sharable Ontologies Library at Stanford University** is an electronic library of public ontologies. Ontologies are specifications of conceptualizations, used to help programs and humans share knowledge. In practice, they consist of a representational vocabulary with axioms by which machine usable semantics are defined. The ontologies in the library are for engineering math and component modeling.

Microkosmos at New Mexico State University and Carnegie Mellon University is another lexicon/ontology system. Microkosmos is a research project aimed at developing a methodology for representing the meaning of natural language texts in a language-neutral interlingual format called a text meaning representation (TMR). TMR's are designed primarily for use in machine translation (MT). Initial work is being performed in the area of lexical-semantic dependency and the basic structure of events and their properties. Future work will focus on aspect, time, modalities, discourse relations, reference, and style.

The Natural Language Software Registry is a collection of information on a wide variety of natural language processing software, both research and commercial. The registry is divided into speech signal analysis, morphological analysis, syntactical analysis, formalisms, semantic and pragmatic analysis, generation, knowledge representation systems, multicomponent systems, NLP-tools, data sets, and applications and text processing. Well over one-hundred systems are included in the registry. Information on availability and price is included for each entry, although some of the information appears to be out-of-date.

Cyc is a large, multi-contextual knowledge base and inference engine. The goal is to construct a foundation of basic common sense knowledge of terms, rules, and relations from every-day life. Currently Cyc contains "tens of thousands of terms" and several dozen assertions about each term. Assertions have all been hand-crafted. In 1995, Cycorp announced that it expected to make a public release of some of the Cyc ontology along with portions of the software. This would be for non-commercial use only, but as of the date of writing, Cycorp has made no further announcement. Commercial availability of Cyc is by means of investing in Cycorp as a partner (at a cost measured in the millions of dollars).

Moby Database is not really a database, rather it is a set of files containing very limited pronunciation, grammatical and synonymy information for a large set of English words. Aside from the limited information content, the entries have a significant number of errors, apparently from faulty data entry.

WordNet is a research system in psycholexicology. WordNet is an online lexical database designed for use under program control. English nouns, verbs, adjectives, and adverbs are organized into sets of synonyms, each representing a lexicalized concept. Semantic relations link the synonym sets. WordNet is composed of around one-hundred thousand words. We will discuss WordNet in detail in the next section.

UMLS is a project being carried out by the National Library of Medicine (NLM) to develop a common reference for medical terminology. Although the bulk of the data in UMLS relates to medical specific terminology, the SPECIALIST lexicon (which is a part of UMLS) contains information on many common English words. In particular, it incorporates information from the two-thousand word core of the Longman's Dictionary of Contemporary English (LDOCE) and from the ten-thousand word base vocabulary of The American Heritage Word Frequency Book. There is no semantic information for the standard English vocabulary, but the level of information relative to syntactic categories is quite comprehensive.

From the above sources, we have selected WordNet and the UMLS SPECIALIST lexicon for use in constructing the CAMA-EDM lexicon. Of the many systems surveyed, WordNet and SPECIALIST have the following attributes to commend them.

- Broad lexical coverage
- Reasonable, consistent, and well documented format
- Sound development methodology
- Continuing development and enhancement
- Free distribution and no royalty fees
- No restrictions on the purposes for which they may be used

None of the other sources seem to meet all these criteria.

As to availability, WordNet (registered trademark of Princeton University) is available by anonymous ftp from clarity.princeton.edu. Included in the data distribution are command-line and XWindows based utilities for searching, browsing and collating data, and extensive documentation and references.

The UMLS Knowledge Sources are available on six CD-ROMS (with documentation) from the National Library of Medicine. It is also available on-line at http://www.nlm.nih.gov/ and the distribution on CD-ROM with hardcopy documentation may soon be eliminated.

In the next section we will discuss WordNet and SPECIALIST in greater detail and describe the methodology by which they have been merged with each other and with the lexical database of military terms created during earlier CAMA programs.

4.2 Methodology

There are no standards for the design and development of lexicons for natural language processing systems. The ARPA sponsored Tipster Architecture program is probably the most serious attempt at designing and implementing standards for modularity and data interchange in natural language systems. The current Tipster requirements are, however, far from complete, and broad implementation is still years away. As a result, neither our data sources nor our target lexicon have any widely accepted format by which to abide. The best we can do is to use sources and create a lexicon which adheres to a reasonable, consistent, and well documented format.

An accurate overview of WordNet and SPECIALIST can probably be best achieved by quoting excerpts from the published literature. The following information regarding WordNet is taken from Miller [18].

> In WordNet, a form is represented by a string of ASCII characters, and a sense is represented by the set of (one or more synonyms that have that sense. WordNet contains more than 118,000 different word forms and more than 90,000 different word senses, or more than 166,000(*f,s*) pairs. Approximately 17% of the words in WordNet are polysemous; approximately 40% have one or more synonyms.
>
> WordNet respects the syntactic categories *noun, verb, adjective*, and *adverb* – the so-called open-class words. For example, word forms like "back," "right," or "well" are interpreted as nouns in some linguistic contexts, as verbs in other contexts, and as adjectives or adverbs in other contexts; each is entered separately into WordNet. It is assumed that the closed-class categories of English – some 300 prepositions, pronouns, and determiners – play an important role in parsing systems; they are given no semantic explication in WordNet.
>
> Inflectional morphology for each syntactic category is accommodated by the interface to the WordNet database. For example, if information is requested for "went," the system will return what it knows about the verb "go." On the other hand, derivational and compound morphology are entered into the database without explicit recognition of morphological relations. For example, "interpret," "interpreter," "misinterpret," "interpretation," "reinterpretation," "interpretive," and "interpretive dancing" are all distinct words in WordNet.
>
> A much larger variety of *semantic relations* can be defined between words and between word senses than are incorporated into WordNet. The semantic relations in WordNet were chosen because they apply broadly throughout English, and because they are familiar – a user need not have advanced training in linguistics to understand them.... WordNet includes the following semantic relations:
>
> a. *Synonymy* is WordNet's basic relation, because WordNet uses sets of synonyms (*synsets*) to represent word senses. Syn-

> onymy(*syn* same, *onyma* name) is a symmetric relation between word forms.
>
> b. *Antonymy* (opposing-name) is also a symmetric semantic relation between word forms, especially important in organizing the meanings of adjectives and adverbs.
>
> c. *Hyponymy* (sub-name) and its inverse, *hypernymy* (super-name), are transitive relations between synsets. Because there is usually only one hypernym, this semantic relation organizes the meanings of nouns into a hierarchical structure.
>
> d. *Meronymy* (part-name) and its inverse, *holonymy* (whole-name), are complex semantic relations. WordNet distinguishes *component* parts, *substantive* parts, and *member* parts.
>
> e. *Troponymy* (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower.
>
> f. *Entailment* relations between verbs are also coded in WordNet.
>
> Each of these semantic relations is represented by *pointers* between word forms or between synsets. More than 116,000 pointers represent semantic relations between WordNet words and word senses....
>
> Polysemy is a major barrier for many systems that accept natural language input.... For example, in computer-assisted instruction, a student asking the meaning of a word should be given its meaning in that context, not a list of alternative senses from which to pick.
>
> WordNet lists the alternatives from which choices must be made. WordNet would be much more useful if it incorporated the means for determining appropriate senses, allowing the program to evaluate the contexts in which words are used. This unmet requirement is a goal for further development.... How best to characterize the contexts associated with word senses remains an open question.

The current work toward providing WordNet with semantic selection based on context is being performed on the basis of statistical studies of word use in large corpora.

The following introduction to the SPECIALIST lexicon is taken from the on-line documentation that is delivered with the UMLS Knowledge Sources.

> The SPECIALIST lexicon has been developed to provide the lexical information needed for the SPECIALIST Natural Language

Processing System (NLP). It is intended to be a general English lexicon that includes many biomedical terms. Coverage includes both commonly occurring English words and biomedical vocabulary. The lexicon entry for each word or term records the syntactic, morphological, and graphemic information needed by the SPECIALIST NLP System.

The lexicon consists of a set of lexical entries with one entry for each spelling or set of spelling variants in a particular part of speech. Lexical items may be "multi-word" terms made up of other words.... Expansions of generally used acronyms and abbreviations are also allowed as multi-word terms. Entries which share their base form and spelling variants, if any, are collected into a lexical record. The base form is the uninflected citation form of the lexical item; the infinitive in the case of a verb; the singular in the case of a noun; and the positive in the case of an inflecting adjective or adverb. The lexical record is a frame structure consisting of slots and fillers. Each lexical record has a base= slot whose filler indicates the base form, and optionally a set of spelling_variants= slots to indicate spelling variants. Lexical entries are delimited by entry= slots filled by the EUI number of the entry. EUI numbers are seven digit numbers preceded by an "E" Each entry has a cat= slot indicating part of speech. The lexical record is delimited by braces (...).

The lexical record for "anaesthetic" given below illustrates some of the features of a SPECIALIST lexical record:

```
{base=anaesthetic
 spelling_variant=anesthetic
 entry=E0008769
        cat=noun
        variants=reg
 entry=E0008770
        cat=adj
        variants=inv
        position=attrib(3)
 }
```

....

Lexical entries are not divided into senses. So, an entry represents a spelling-category pairing regardless of semantics. The noun "act" has two senses both of which show a capitalized and lower case spelling; an act of a play and an act of law. Since both senses share the same spellings and syntactic category, they are represented by a single lexical entry in the current lexicon. The verb "act" is not included in this lexical record since it has no capitalized spelling variant....

When different senses have different syntactic behavior, codes for each behavior are recorded in a single entry. For example, "beer" has two senses: the alcoholic beverage and the amount of a standard container of that beverage.

Currently the SPECIALIST lexicon contains about 65,000 records.

Words are selected for lexical coding from a variety of sources. Around 20,000 words from the UMLS Test Collection of MEDLINE abstracts together with words which appear both in the UMLS Metathesaurus and Dorland's Illustrated Medical Dictionary form the core of the words entered. In addition, an effort was made to include words from the general English vocabulary. The 10,000 most frequent words listed in The American Heritage Word Frequency Book and the list of 2,000 words used in definitions in Longman's Dictionary of Contemporary English have been also coded. Since the majority of the words selected for coding are nouns, an effort has been made to include more verbs and adjectives by identifying verbs in current MEDLINE citation records, by using the Computer Usable Oxford Advanced Learner's Dictionary, and by identifying potential adjectives from Dorland's Illustrated Medical Dictionary using heuristics developed by McCray and Srinivasan (1990).

A variety of reference sources are used in coding lexical records. Coding is based on actual usage in the UMLS Test Collection, dictionaries of general English, primarily learner's dictionaries which record the kind of syntactic information needed for NLP, and medical dictionaries. Longman's Dictionary of Contemporary English, Dorland's Illustrated Medical Dictionary, Collins COBUILD Dictionary, The Oxford Advanced Learner's Dictionary, and Webster's Medical Desk Dictionary were used.

The basic sentence patterns of a language are determined by the number and nature of the complements taken by verbs, since the complementation of the main verb largely determines the structural skeleton of a sentence. SPECIALIST recognizes five broad complementation patterns: intransitive, transitive, ditransitive, linking and complex-transitive. These complementation classes are manifested in the lexicon as slots filled by codes further specifying the verbs complementation pattern. Table 4.2-1 indicates the slot name associated with each complementation class.

Class	Slot Name
intransitive	intran
transitive	tran=
ditransitive	ditran=
linking	link=
complex-transitive	cplxtran=

Table 4.2-1. Verb Complementation Patterns in SPECIALIST

Intransitive verbs are those which can appear with no complements at all....

Transitive verbs take a single object complement. This complement may be a noun (direct object), a prepositional phrase, a finite complement etc... Ditransitive verbs have more than one object complement.... Verbs can, and often do, fall into more than one complementation class....

Verb entries also encode each of the inflected forms, (principal parts of the verb) in a variants= slot. Verbs are inflectionally classified as regular, Greco-Latin regular or irregular....

As described above noun entries describe the inflection of the nouns (pluralization) in a variants= slot and spelling variation in a spelling_variant= slot. The compl= slot indicates complementation for nouns. A nominalization= slot indicates that the noun is the nominalization of a verb or adjective.

In addition to inflection (variants=) codes and complement codes, adjectives in SPECIALIST have position codes, in a position= slot, to indicate the syntactic positions in which they occur. Adjectives that occur pre-nominally in noun phrases are marked attrib(), in the position= slot. The numerical argument of the attrib() slot indicates where in the normal sequence of noun premodifiers this adjective occurs. Qualitative adjectives (attrib(1)), normally precede color (attrib(2))and classifying (attrib(3)) adjectives.

Adjectives that can occur in predicate adjective constructions, have the code pred in their position= slot, and adjectives which can occur post nominally have the code post....

Adverbs in SPECIALIST are coded to indicate their modification properties in a modification_type= slot. SPECIALIST recog-

> nizes sentence, verb-phrase and intensifier type adverbs, as well as classifying verb-phrase and sentence adverbs into manner, temporal and locative types. Adverbial particles like "up"... are also listed as adverbs in SPECIALIST, with a modification_type indicating that it is a particle.

The grammar and parser for SPECIALIST are still under development and not yet available to the public. For this reason, the lexicon is also subject to change and expansion, but may be considered reasonably stable in format.

The SPECIALIST lexicon provides the grammatical information necessary to the CAMA front end, but is short on semantic information and contains mostly medical and scientific terms. The WordNet lexicon contains semantic information about a broad range of terms, but contains little grammatical information and only has nouns, verbs, adjectives, and adverbs.

We decided to base our lexicon on the SPECIALIST lexicon, and prune the medical terms by intersecting it with the WordNet lexicon. By removing the nouns, verbs, adjectives and adverbs that do not appear in the general use WordNet lexicon (90,000 words), we trimmed the SPECIALIST lexicon from 65,000 down to 24,000 core records. (The resulting lexicon is slightly skewed to medical and scientific terms, since the huge WordNet database contained words specific to many different fields.) Additionally, we added fields to the SPECIALIST records to represent the WordNet semantic information, which will be integrated at a later date.

A sample record from the resulting lexicon is show in Figure 4.2-1. This generally follows the SPECIALIST format. The synset, hypernym, gloss, and pertainym fields come from the WordNet definitions. The variants, position, and stative fields, and many others are native to the SPECIALIST records, and this information is essential to the grammar development.

These large records will be useful in later versions of CAMA, but another set of records supplied by the SPECIALIST lexicon provides enough information for the CAMA-EDM project. The UMLS package includes ASCII relational databases that reference the entry numbers of the SPECIALIST records. One such database is the agreement and inflection table (*lragr*), which is helpful in determining agreement between elements of a sentence. It includes seven fields:

a. EUI – the unique SPECIALIST entry number

b. STR – a form string of the base word

```
{base=American
entry=E0000248
        cat=noun
        synset= American
        hypernym= inhabitant, dweller, denizen
        gloss= (a native or inhabitant of the United States)
        synset= American English, American language, American
        hypernym= English, English language
        gloss= (the English language as used in the US)
        synset= American
        hypernym= native
        gloss= (a native or inhabitant of America)
        variants=reg
        proper
entry=E0000249
        cat=adj
        synset= American
        pertainym= Pertains to noun -> America (Sense 1)
        gloss= (of or relating to the United States of America
                or its people or language or culture:
                "American citizens"; "American English";
                "the American dream")
        synset= American, american
        pertainym= Pertains to noun -> America (Sense 2)
        gloss= (of or relating to or characteristic of the
                continents and islands of the Americas: "the
                American hemisphere"; "American flora and fauna")
        variants=inv
        position=attrib(3)
        position=pred
        stative
}
```

Figure 4.2-1. Sample record from merged lexicon

c. SCA – the syntactic category (part of speech) of the base word

d. TNS/CNT/CMP – the tense (TNS – for verbs), or the count or uncount status (CNT – for nouns) or positive, comparative, or superlative (CMP – for adjectives and adverbs)

e. PRS/PER – first, second or third person (PRS), or periphrastic (PER – for adjectives and adverbs)

f. NUM – number (for nouns, verbs, and determiners)

g. BAS – base or citation form of the entry

These fields are enough for the unitizer and symbolizer. In particular, the syntactic category, the tense, and the base are used to identify different types of tokens used in the CAMA-EDM grammar. In fact, since the *lragr* includes forms of words, the morphological and inflectional information are provided by directly looking up the string found in the text. So the *lragr* that corresponds to the pruned SPECIALIST lexicon is used as the lexicon for CAMA-EDM.

Furthermore, we incorporated vocabulary that is specific to Army logistics teleconference messages. These words were culled from nearly two-hundred sample teleconference messages on a prior CAMA program. The database for these words includes inflected and base forms of each word, part-of-speech, expansions of acronyms, and English definitions. We transformed this database into the *lragr* format and added these records to our lexicon.

The WordNet *-onymy* relations provide information that will be useful when semantic meaning is added to CAMA. Most notably, these relations will aid in normalizing text to a standard collection of concepts. For example, imagine that a user wants to extract information on the transport of weapons. The CAMA lexicon would need to recognize which words indicated types of movement and which words were a type of weapon. The hyponymy and hypernomy relations provide exactly this. A sidewinder is a hyponym (example) of weapon, and a missile is a hyponym of weapon. Conversely, weapon and missile are hypernyms of sidewinder. So CAMA will have to extract all documents that contain sentences describing a hypernym of transport occurring to a hypernym of weapons. Synonyms will be detected similarly, as well as sentences that describe the negation of the antonym of a particular word (e.g. "The missiles will not stay at their current location"). Thus a range of words will normalize to the "concept" described in an extraction specification by the use of relations currently present in the WordNet database.

4.3 Principles

We have noted that the WordNet *-onymy* relations provide some level of semantic information. This information is primarily of the type that can be considered the *sense* of the words. That is, how the words relate to each other either as synonyms, antonyms, hyponyms, hypernyms, and troponyms. These relations do not help directly with the semantic aspect of *reference* – that is, referencing some entity by means of description.[1] The only relations in WordNet that relate to reference are meronymy and holonymy. Determination of the referent of a polysemous word may be possible based on meronymy. Consider for example the sentence "During disassembly the x was removed from the y." If $x =$ "pan" and $y =$ "engine" and our system knows "pan" either as "a self-contained baking utensil" or "a part-of an engine that holds oil" then the proper reference can be selected. There are, however, many other relations than meronymy and holonymy that are needed to make referential selection.

The relations required for referential selection can be coded in at least two ways – direct and indirect. Direct coding requires that all the necessary relations be identified, defined and that each entry in the lexicon have the appropriate definitions assigned. Obviously, this is a huge task that is complicated by the fact that ontology (the study of the nature of being) is not well understood and is certainly not standardized. Most ontologies that are used for computational language processing systems (*e.g.* Microkosmos and the UMLS Semantic Network) are admittedly *ad hoc* and may be of little use even in applications closely related to the original. Construction of a "complete" ontology of common-sense knowledge as in the Cyc project has occupied many people for well over a decade with results that are still far from the original objectives. This would seem to indicate that the direct approach to coding the relations required for referential selection should be kept to a minimum as required by the application at hand.

The indirect approach uses the relations which we have associated with *sense*. This approach assumes that particular referents can only reasonably stand in particular relations to other referents that fall into a particular class. This approach requires that each different meaning for a word be classified appropriately according to both sense and grammatical relations. WordNet provides an extensive classification on the basis of sense relations but provides grammatical classifications only at a course level. WordNet does not, for instance, indicate that "hope" and "believe", while similar in meaning and both being verb, have different grammatical realizations.

The grammatico-semantic classification scheme by which we have been working is due

[1] The terms *sense* and *reference* are not used consistently in the literature. The definitions given here are in force only over the scope of this subsection.

to Dixon [6]. Dixon has proposed a grammar in which meanings in the major classes of verb, noun, adjective and adverb are further classified according to syntactic function and then again by the nature of the semantic roles in which they can stand or which they can stand in relation to.

It is not easy to capture the essence of Dixon's work in a brief space without either over simplification or making it sound too much like the generative tradition in linguistics which is more commonly known. Consider, however, that one class of verbs is the Primary-A type - verbs that have roles filled by noun phrases that have heads of the Concrete type. Among the Primary-A verb types are Affect-g and Affect-h types which we will use for examples. Affect-g, the Build subtype, refers to manufacture and cooking. These verbs involve an Agent creating something called the Product, possibly via the intermediary of a Manip. Affect-h, the Break subtype, involves an Agent causing some object (Breaking role) to loose its physical unity, possibly via the intermediary of a Manip. And so, we can have "Mary(Agent) built(Primary-A/Affect-g) a wall (Product) with those bricks(Manip)." or "Mary(Agent) broke(Primary-A/Affect-h) the vase(Breaking) with that stick(Manip)." The use of semantic roles allows us to specify what types or classes of referents can fill a role, they define the secondary relations that can be expressed relative to the roles, and they add specificity to the understanding of the referents once the roles are assigned.

A considerable amount of further explanation is needed, but this is not the place for it. We mention the use of semantic roles here only because of the effect on lexicon and grammar development. The representation and manipulation of semantic roles has a much greater part in relation to the CCLS back-end semantic processor which is not properly a part of the CAMA-EDM program. For this reason we will not attempt to go into further detail on this subject.

In summary, the CAMA-EDM lexicon draws from three source lexicons, the UMLS SPECIALIST lexicon, WordNet, and the CAMA lexicon of teleconference vocabulary. SPECIALIST and WordNet were intersected taking syntactic information from SPECIALIST and word sense information from WordNet. The CAMA lexicon was unioned with the intersection of SPECIALIST and WordNet to produce a lexicon of about 24,400 words. The grammar and semantic roles from Dixon's semantic grammar of English have been defined in a machine readable format as the basis for the identification of particular referents of polysemous words. In all, a lexicon of considerably broader coverage than what was contractually stipulated has been produced.

5 EXPERIMENT OBSERVATIONS AND EVALUATION

In order to evaluate the work done under the CAMA-EDM program, we have thoroughly tested the system, run experiments with certain system modules, and researched other approaches to the text processing problem. The system testing is documented in the Software Test Plan [13]. Here we describe the experiments run on the multi-phase chart parser in Section 5.1 and the research into other work in Section 5.2.

5.1 Evaluation of the Multi-phase Chart Parser

The symbolizer includes a multi-phase chart parser. Multi-phasing means that a grammar may be broken into a sequence of stages (called phases). Each phase operates on set of tokens, with the new nonterminals of an earlier phase used as terminals of the subsequent phase. Each phase has its own parse table. (See Section 3.2 for a description of chart parsing). After the unitizer creates the initial tokens, they are put on a chart and parsed using the first phase parse table. When the chart parser has finished, the tokens that do not appear as terminals in any future phase are removed from the chart. Then the chart is parsed again, using the second phase parse table. This continues through all of the phases of the grammar.

Breaking down a grammar into many phases improves the readability of the grammar. Most grammars contain dozens of rules, which make them difficult to analyze if taken as a whole. By separating the rules into many phases, a grammar becomes easier to examine and analyze. This modularity also allows a separation of the core grammar from rules that users may need to modify. Users can modify particular phases, depending upon the organization of the grammar, without changing the core functionality of the grammar.

More importantly, multi-phasing increases the efficiency of the symbolizer. We conducted tests of the system using two different grammars (See Tables 5.1-1 and 5.1-2). One grammar had four phases of rules, and the other had one phase using all of these rules. We ran them on sentences that would match a full sentence rule: some sentences were highly ambiguous, and others were uniquely parseable. In all tests, the parser using the multi-phase grammar executed more quickly than the parser using the single phase grammar. Furthermore, each parser produced exactly the same list of parses for each of the sentences (including the test in which there were 83808 possible parses).

Key to multi-phasing is the removal of the useless tokens between phases. The symbolizer keeps track of which tokens are used by each phase. If tokens of a particular type are not used in any rule beyond a certain phase, then all of those tokens may be removed

		Reductions	
Description	Parses	Single Phase	Multiphase
2 unambiguous sentences	1	94	86
4 unambiguous sentences	1	181	139
24 unambiguous sentences	1	2346	1744
48 unambiguous sentences	1	7716	5650
1 ambiguous sentence	36	353	203
3 ambiguous sentences	32	1652	1526
5 ambiguous sentences	792	6297	3123

Table 5.1-1. Comparison of single phase and multi-phase reductions

	Single Phase			Multiphase		
Parses	Reducs	Real Time	CPU Secs	Reducs	Real Time	CPU Secs
2	47	-	-	32	-	-
8	138	-	-	84	-	-
36	446	-	-	221	-	-
168	1774	-	-	754	-	-
792	7916	-	-	3149	-	-
3744	36814	0:17	15.0	14332	0:07	6.0
17712	173348	1:32	75.0	67055	0:36	32.0
83808	819130	59:58	361.0	16258	9:55	139.0

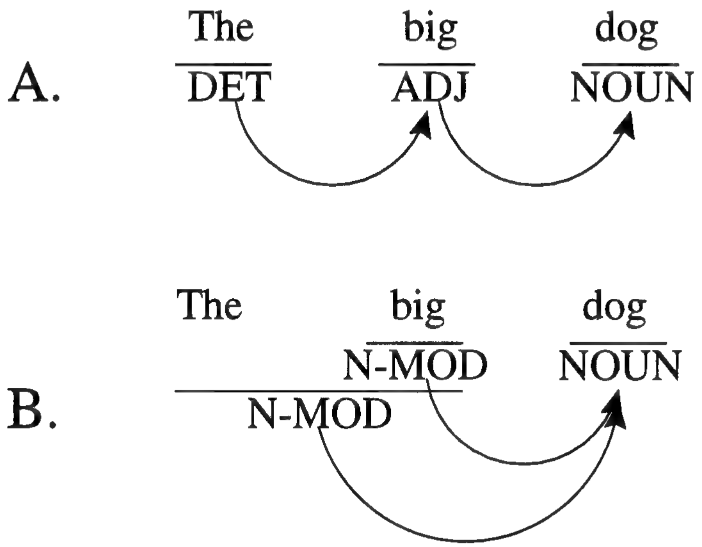
Each line of the table represents one sentence processed using a single phase grammar and a multi-phase grammar. The first sentence is of the form “X active”. Each subsequent test added “over Y” to the sentence, e.g. the third is “X active over Y over Z”. Time of processing below 10000 reductions was negligible. Also note that there were approximately 2300 reductions per CPU second for both grammars.

Table 5.1-2. Single phase versus multi-phase execution times

after that phase. This results in a much smaller chart.

An obvious advantage of removing tokens is that the parser will not need to traverse a chart full of useless tokens, indicating that time efficiency may improve. In all of our tests, however, the relative time per reduction was approximately identical. In fact, the parser had to traverse the chart more times using the multi-phase grammar. Using the single phase grammar, all of the rules are available at the beginning of the parse, and so the parser can check for matches at all times.

The significant advantage of removing useless tokens is that less reductions are produced. Removing tokens orphans other tokens, which prevents the symbolizer from noticing some spurious pattern matches.



(A) shows the chart before phase 1, and (B) shows the chart after phase 1.

Figure 5.1-1. An example of orphaning

Consider the phrase “the big dog”. (See Figure 5.1-1). Say that the first phase of a grammar has rules for creating noun modifier tokens. These rules match a determiner (such as “the”) followed by any number of adjectives, or just any number of adjectives. So “the big” and “big” will be made into noun modifiers (or N-MODs, see Figure 5.1-1 B). Now, say that the second phase makes noun phrases from noun modifiers and nouns, but has no mention of determiners and adjectives. The determiners and adjectives will be removed from the chart. The noun phrase rule has two possible patterns to match – “the big” with “dog”, and “big” with “dog”. But “big dog” will never be made, since the "big" noun modifier is *orphaned.* The symbolizer starts at the beginning of the chart, and follows links to the end. When the "the" token was present, the symbolizer could find the "big" token by itself. But now, since "the" has been filtered, the symbolizer cannot reach "big". So only "the big dog" is turned into a noun phrase.

A single phase grammar will locate the "big dog" combination and make the reduction, since there is no token filtering. But since we have a better path, i.e. "the big dog", "big dog" does not need to be made. We will only orphan when a larger token is available. If "the big" was not made, an "unknown" token would replace "the" when it is removed. The unknown token would point to the "big" noun modifier, and the multi-phase grammar would in fact create the "big dog" noun phrase. The multi-phase grammar will never miss a full parse of the text if one exists.

In summary, there are two major advantages to multi-phasing. First, dividing the grammar into phases makes grammar maintenance easier. Second, multi-phasing orphans tokens by the filtering process. This filtering prevents useless reductions when better reductions are available. We believe that most of the efficiency improvements we observed were due to this phenomenon. Time is saved, since executing reduction code takes a large portion of the parsers time. Space is saved, since new tokens do not need to be created and kept in memory.

5.2 Comparison to Other Work

CAMA represents an application of natural language processing (NLP) technology. There has been significant work on NLP for extraction of information from military or similarly structured messages. Much of the work in this area has been showcased in the Message Understanding Conferences (MUCs) sponsored by DARPA. The latest conference, MUC-6, was held in Columbia, Maryland in November, 1995. These conferences have shown a steady, yet slow, increase in overall performance, measured by recall and precision. The CAMA front end modules, the unitizer and symbolizer, are similar to components making up several of the MUC systems. The key difference is that in CAMA-EDM we have developed modules which can be placed in operational systems. Therefore, robustness and maintainability were priorities. In this section we compare the CAMA system to others that have been demonstrated in the MUCs. Much of the information reported here was gathered through the Defense Technical Information Center (DTIC). A search of technical reports was conducted using the Scientific and Technical Information Network (STINET) via the World Wide Web.

The organization of the CAMA front end is typical of most text extraction systems.

The unitizer breaks the text into segments, often referred to as text zoning, and tokenizes each of the segments, assigning a category to each word. The symbolizer accepts the tokens and performs a grammatical analysis. Like the Lockheed Martin (formerly GE) NLToolSet, the unitizer recognizes structured text such as dates, times, and coordinates. Through the use of parallel lexical analysis, CAMA can detect these structured pieces of text within formatted or free text. NLToolSet also has a module for identifying proper names. CAMA will recognize proper names that are part of its lexicon. The symbolizer works similarly to the FASTUS system produced by SRI. In FASTUS, the extraction process is performed using a cascade of finite state automata. Each stage matches patterns of word or phrase categories. The patterns at each successive stage match larger pieces of text. The output of the final stage is a set of structures which form a summary of the extracted information. The shift-back chart parser is the engine of the symbolizer and works similarly to a single phase of FASTUS. Like FASTUS, the chart parser will run in multiple phases. However, while FASTUS is statically configured to a set number of stages, the symbolizer is easily reconfigured to any number of phases. We believe the multi-phase chart parser to be unique to the CAMA system.

A principle feature of the CAMA front end is its implementation style. Most of the systems which were tested in the MUCs have been laboratory prototypes implemented in Lisp or Prolog, NLToolSet being an exception. Recently, some of the top performing systems have begun reimplementation into C or C++. Both FASTUS from SRI and the CIRCUS system from the University of Massachusetts at Amherst are currently being converted from Lisp. The CAMA system has been designed from the ground up using an object-oriented methodology and is implemented fully in C++. In addition, a rigorous engineering methodology has been enforced throughout development. The goal has been software which can move beyond a laboratory prototype into an operational environment.

An advantage held by systems such as FASTUS, CIRCUS, and NLToolSet is the size and maturity level of the lexicons they use. Through participation in the MUCs, each of these systems has undergone significant testing against a government supplied set of documents. To effectively participate in the MUCs, the content and structure of their lexicons has been developed over several years. To make up for this difference, our lexicon development for CAMA has leveraged off other work which is publically available. This includes WordNet from Princeton University and UMLS from the National Library of Medicine. To maximize the effectiveness of CAMA within the logistics domain, vocabulary culled from a sample set of teleconferencing messages has been included in the lexicon. This approach supports the eventual insertion of CAMA into an operational environment. Lexicon research and development is explained in detail in Section 4.

Another distinguishing characteristic of CAMA is the preservation of ambiguity throughout processing. The front end modules have been designed to eventually be integrated with a text understanding function. To make this possible, both the unitizer and symbolizer encode and retain multiple ambiguous, although valid, representations of text. Alternate representations are filtered out only when either the unitizer or symbolizer have enough information to determine the correct interpretation. Otherwise, the ambiguity is passed along to the next stage of processing. An example of ambiguity in the unitizer is discussed in Section 3.1. It is difficult to directly compare this feature to other systems except to say that retaining ambiguity requires small and efficient data structures for reasonable memory and speed performance. A system implemented in Lisp or Prolog may have difficulty meeting this requirement.

System maintainability has been key to the design of the unitizer and symbolizer. To make the system modifiable, we have cleanly divided the data from the source code. The data which controls the behavior of the system is represented in specification files. A type of specification file is also used in FASTUS to define grammar rules. For CAMA, simple languages have been developed for use in specification files. Modifying specification files will require experience with documents to be analyzed and knowledge of syntax and semantics of the specification language. While this approach requires more expertise than is required on a trainable system such as CIRCUS, the technology for automatically training an extraction system has not reached a level of maturity suitable for operational systems.

To summarize, CAMA, although similar in many ways to other information extraction systems, has four distinguishing characteristics. First, the parallel lexical analysis and multi-phase chart parsing techniques used in the front end modules, we believe, are unique to CAMA. Second, a system designed according to object-oriented principles and implemented using sound engineering has produced software which can be directly inserted into an operational prototype. The third characteristic is the functionality, specifically the preservation of ambiguity, necessary for eventual integration with a text understanding module. Lastly, the inclusion of vocabulary from sample teleconferencing messages has enhanced the utility of CAMA within the logistics domain.

6 ACCOMPLISHMENTS AND CONCLUSIONS

The CAMA-EDM program has continued the research and development of a natural language processing system for Army logistics messages. There were two primary goals for this effort: first, develop a robust infrastructure for doing tokenization and parsing, also referred to as the CAMA front end, and, second, continue the research into a lexicon for use with the L-space model. This program is a milestone on the way to the ultimate goal of a text processing system which automatically extracts entities and actions of interest, links cause with effect, and recognizes trends. Past CAMA programs had shown the viability of the L-space model for doing the level of text understanding necessary to reach these goals. The CAMA-EDM program has produced robust, well engineered software modules for doing text segmentation, tokenization, and parsing. In addition, lexicon research has defined a semantic structure for lexical entries and produced a large lexicon for use with the parser. The following summarizes the major accomplishments of the program.

- CAMA has been implemented according to the CoPE architecture [19], dividing the system into on-line, off-line, and control file subsystems.

- A technology for text unitization called em parallel lexical analysis was developed. Parallel lexical analysis supports modularity in both specification and operation of the unitizer.

- The chart parser developed under previous CAMA programs was enhanced. The new chart parser is more robust, modular, and supports multi-phase operation. Multi-phasing increases the efficiency of chart parsing.

- Off-line compilers were written to translate the specification files into binary objects. Each of the three types of specification files; unitizer, symbolizer, and L-space; has an off-line compiler. These compilers perform complete error checking on the specification files.

- Lexicon research combined two publicly available lexicons, WordNet and UMLS SPECIALIST, along with vocabulary from sample teleconferencing messages to produce a lexicon with 24,400 entries.

- A representation for semantic information in lexical entries was forged from an intersection of WordNet and UMLS SPECIALIST.

- A new demonstration user interface was created using Netscape Navigator. This interface visually presents the internal processing of the on-line system and allows immediate inspection of system results for any message.

- All software has been designed using an object-oriented methodology and implemented in C++. All source code files have been maintained under configuration control.
- A presentation describing the technology used in the CAMA front end was made at the Symposium for Advanced Information Processing and Analysis [20].
- The system has been tested against nearly 200 sample teleconferencing messages.
- A commercial product, currently being used in medical data processing, has spun off from this work.

The CAMA-EDM program has surpassed expectations in two important areas. First, the size of the resulting lexicon, 24,400 entries, is much larger than was planned. Leveraging from existing, publicly available lexicons has allowed us to produce a lexicon with broad coverage. A lexicon of the current size would be sufficient for many operational environments. The second area is the quality of the demonstration user interface. The Netscape interface presents the results in a way which is simple, visual, and interactive. This allows CAMA to be more effectively demonstrated to audiences with either a technical or operational view. Overall, this program has produced a usable and demonstrable prototype, showing how state-of-the-art text processing technology can be effectively engineered and applied to real-world problems.

Looking forward to meeting the ultimate goals of CAMA, we feel the L-space model has the potential for superior performance in text understanding. L-space is a model of information representation based on lattice theory [4, 22]. The design of L-space and the entire CAMA system was inspired by work in cognitive linguistics [9, 16, 17]. What our approach attempts to do is implement cognitively-based information processing using principled methods [2]. We have developed and implemented the CAMA front end according to this view. The behavior of the front end modules is well defined and easily modifiable, displaying a high degree of technical refinement. The intention is to continue this same style in the development and implementation of L-space.

7 RECOMMENDATIONS

CAMA demonstrates NLP technology applied to the domain of Army logistics. To move this technology from the laboratory to the field requires the identification of an appropriate real-world application. Such an application should have a clear expectation that CAMA technology would add value as well as a defined concept of operations. This would guide the next stage of development and focus the text understanding capabilities into areas which would provide the most benefit. CAMA is now a usable, demonstrable technology. Fitting it into an operational environment is the next challenge.

In its current configuration, CAMA is a text extraction system, identifying entities and artifacts within teleconferencing messages. However this technology has several other potential uses. A commercial version of the software is currently being used as a generator of metadata, providing the key values necessary to build a database. Text retrieval based on concepts, instead of keywords or patterns, is another potential application. Concept-based retrieval would find documents of interest by matching semantic representations of the query against the documents. Such a system would require further development of L-space technology. A third potential application is document routing. The current technology can be used to recognize different types of documents based on text structure and phrase patterns. This criteria could be extended to include the conceptual content of the text.

A significant accomplishment of the CAMA-EDM program is the new user interface which uses HTML files viewed through Netscape Navigator. This interface presents both the internal processing of the on-line system as well as the final results in a visual and interactive way. An operator's interface could be modelled after this prototype and use Netscape as the software platform. This would allow results to be viewed on any machine which supports a version of Netscape Navigator. This strategy lowers costs and would simplify the insertion of CAMA technology into an operational environment.

8 REFERENCES

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley, Reading, PA, 1986.

[2] Daniel M. Davenport. Cognitive Information Processing. Technical Memo, Center for Intelligent Information Processing, University Park, PA, September 1995.

[3] Daniel M. Davenport and Mark L. Morsch. CoPE, A Cognitive Processing Engine. In *Proceedings of the AIPA95 Symposium*, Tyson's Corners, VA, March 1995.

[4] B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, England, 1990.

[5] Defense Advanced Research Projects Agency, McLean, VA. *Proceedings of the Fourth Message Understanding Conference (MUC-4)*, June 1992.

[6] R.M.W. Dixon. *A New Approach to English Grammar on Semantic Principles.* Clarendon Press, Oxford, 1991.

[7] Charles J. Fillmore. The Mechanisms of Construction Grammar. *BLS*, 14:35–55, 1988.

[8] Adele E. Goldberg. *Constructions.* University of Chicago Press, Chicago, Ill., 1995.

[9] Daniel T. Heinze. *Computational Cognitive Linguistics.* PhD thesis, The Pennsylvania State University, University Park, PA, 1994.

[10] Daniel T. Heinze and Daniel M. Davenport. Crisis Action Message Analyzer – EDM. In *Proceedings of the IEEE Dual Use Technologies and Applications Conference*, Utica, NY, May 1995.

[11] Jerry R. Hobbs, Douglas E. Appelt, John S. Bear, David J. Israel, and W. Mabry Tyson. FASTUS, A System for Extracting Information from Natural Language Text. Technical Note No. 519, SRI International, Menlo Park, CA, November 1992.

[12] HRB Systems, Inc. and The Pennsylvania State University – Center for Intelligent Information Processing, University Park, PA. *A Proposal for Crisis Action Message Analyzer EDM*, March 1995.

[13] HRB Systems, Inc. and The Pennsylvania State University – Center for Intelligent Information Processing, University Park, PA. *Software Test Plan for the Crisis Action Message Analyzer EDM*, June 1996.

[14] HRB Systems, Inc. and The Pennsylvania State University – Center for Intelligent Information Processing, University Park, PA. *Software User's Manual for the Crisis Action Message Analyzer EDM*, June 1996.

[15] Mark A. Jennings and Daniel M. Davenport. ConceptCrawler. In *Proceedings of the AIPA96 Symposium*, Tyson's Corners, VA, March 1996.

[16] Ronald W. Langacker. *Foundations of Cognitive Grammar, vol I.* Stanford University Press, Stanford, CA, 1987.

[17] Ronald W. Langacker. *Foundations of Cognitive Grammar, vol II.* Stanford University Press, Stanford, CA, 1991.

[18] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(14):39–41, November 1995.

[19] Mark L. Morsch. CoPE, A Cognitive Processing Engine. Technical Memo, Center for Intelligent Information Processing, University Park, PA, October 1995.

[20] Mark L. Morsch. Elements of Programmable Text Extraction. In *Proceedings of the AIPA96 Symposium*, Tyson's Corners, VA, March 1996.

[21] Rome Laboratory, Rome, NY. *Revised Statement of Work for Crisis Action Message Analyzer*, July 1995. PR No. I-5-4181.

[22] Dana Scott. Outline of a Mathematical Theory of Computation. Technical Monograph PRG-2, Oxford University Computing Laboratory, Programming Research Group, Oxford, England, 1970.

[23] SunSoft and Rogue Wave Software, Inc., Mountain View, CA. *Tool.h++ Introduction and Reference Manual*, August 1994.

[24] Masaru Tomita. *Current Issues in Parsing Technology.* Kluwer Academic Publishers, Boston, MA, 1991.

APPENDIX A

10 TERMS AND ABBREVIATIONS

Tables A-1 and A-2 define relevant acronyms and abbreviations. Table A-3 defines terms which are useful for this program.

Acronym	Meaning
C++	"C++" programming language
CAMA	Crisis Action Message Analyzer
CCLS	Computational Cognitive Linguistics System
CDRL	Contract Data Requirements List
CIIP	Center for Intelligent Information Processing
CoPE	Cognitive Processing Engine
CSCI	Computer Software Configuration Item
CSC	Computer Software Component
CSU	Computer Software Unit (also called a "unit")
DARPA	Defense Advanced Research Projects Agency
EDM	Exploratory Development Model
FSF	Free Software Foundation

Table A-1. Acronyms and Abbreviations

Acronym	Meaning
HTML	HyperText Markup Language
IR	Intelligence/Reconnaissance Directorate of RL
IRAA	Speech Processing Branch of IR
LIA	US Army Logistics Integration Agency
MUC	Message Understanding Conference
NLP	Natural Language Processing
NLU	Natural Language Understanding
RCS	Revision Control System
RL	Rome Laboratory of USAF
SOW	Statement Of Work
SPARC	Scalable Processor ARChitecture (as in SPARCstation)
SRS	Software Requirements Specification
STD	Software Test Description
SUM	Software Users Manual
UMLS	Unified Medical Language System
USAF	United States Air Force

Table A-2. Acronyms and Abbreviations (cont'd.)

Term	Meaning
Chart Lexing	The technique used in the unitizer for performing ambiguous lexical analysis.
Chart Parsing	An approach to grammatical analysis used in the symbolizer which handles part-of-speech ambiguity.
Cognizer	Third stage of CAMA; its principle function being elaboration.
Extraction	Recognition and reporting of the important information in a document.
L-space	Semantic model used in CAMA based on lattice theory.
Parser Generator	Tool used in building a parser.
Recognizer	Fourth stage of CAMA; its principle functions being categorization and interpretation.
Regular Expression	A string representing a pattern of characters.
Segmentation	Operation which identifies the major pieces of a document.
Symbolizer	Second stage of CAMA performing the grammatical analysis.
Tokenization	Operation that breaks text segments into lexical units; accomplished in CAMA using chart lexing.
Unitizer	First stage of CAMA containing the segmentation, tokenization, and document identification operations.

Table A-3. Useful Terms

☆U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-61066